# Fast Parallel Molecular Solutions for DNA-based Computing: the Subset Product Problem

Sientang Tsai[1], Wei-Yeh Chen[2]
*Department of Information Management*
*Southern Taiwan University, Yuan Kung distr. ,Tainan city, Taiwan, R.O.C*
[1]E-mail: tsai@mail.stut.edu.tw
[2]E-mail:cwycwy@mail.stut.edu.tw

*Abstract*—**It is shown first by Adleman that deoxyribonucleic acid (DNA) strands could be employed towards calculating solution to an instance of the NP-comp lete Ham iltonian Path Problem (HPP). Lipton also de monstrated that Adle man's te chniques could be used to solve the satisfiabilit y (SAT) problem . In t his paper, it is demonstrated ho w the DNA operations presented by Adleman and Lipton can be us ed to develop the DNA- based algorithm for solving the Subset Product Problem.**

***Keywords***—Biological Computing, Molecular Computing, DNA-based Computing,NP-complete Problems, Subset Product Problem.

## I. INTRODUCTION

In 1961, Feynman first offered bio-molecular computation, but his idea was not implemented by experiments for a few decades [1]. Adleman [2] in 1994 succeeded to solve an instance of the Hamiltonian path problem in a test tube, just by handling DNA strands. From [5], it was indicated that an optimal solution of every NP-complete or NP-hard problem is determined from its characteristics. DNA-based algorithms had been proposed to solve many computational problems and those consisted of the satisfiability problem [3], the maximal clique [6], three-vertex-coloring [7], the set-splitting problem [8],the dominating-set [9], the maximum cut problem [10] and the binary integer programming problem[11]. One potentially significant area of application for DNA algorithms is the breaking of encryption schemes [12]-[13]. From [14]-[15], DNA-based arithmetic algorithms are proposed. Furthermore from [16], DNA-based algorithms for constructing DNA databases are also offered. The intent of this work is to apply the biological operations in the Adleman-Lipton filtering model to develop the DNA-based algorithms for solving the subset product problem.

The rest of the paper is organized as follows. Section II introduces DNA models of computation proposed by Adleman and his co-authors in details. Section III introduces the DNA program to solve the subset product problem from solution spaces of DNA strands. Conclusions are drawn in Section IV.

## II. DNA MODELS OF COMPUTATION

In the last decade there have been revolutionary advances in the field of biomedical engineering particularly in recombinant DNA and RNA manipulating. Due to the industrialization of the biotechnology field, laboratory techniques for recombinant DNA and RNA manipulation are becoming highly standardized. Basic principles about recombinant DNA can be found in [20]-[24]. In this subsection we describe eight biological operations useful for solving the subset product problem. The method of constructing DNA solution space for the subset product problem is based on the proposed method in [18]-[19].

A (test) tube is a set of molecules of DNA (a multi-set of finite strings over the alphabet {*A*, *C*, *G*, *T*}). Given a tube, one can perform the following operations:

1. *Extract*: Given a tube *T* and a short single strand of DNA, "*s*", produce two tubes +(*T*, *s*) and –(*T*, *s*), where +(*T*, *s*) is all of the molecules of DNA in *T* which contain the strand "*s*" as a sub-strand and –(*T*, *s*) is all of the molecules of DNA in *T* which do not contain the short strand "*s*".
2. *Merge*: Given tubes $T_1$ and $T_2$, yield $\cup(T_1, T_2)$, where $\cup(T_1, T_2) = T_1 \cup T_2$. This operation is to pour two tubes into one, without any change of the individual strands.
3. *Amplify*: Given a tube *T*, the operation Amplify (*T*, $T_1$, $T_2$), will produce two new tubes $T_1$and $T_2$ so that $T_1$ and $T_2$ are a copy of *T* ($T_1$ and $T_2$ are identical) and *T* becomes an empty tube.
4. *Append*: Given a tube *T* and a short strand of DNA," *s*", the operation will append the short strand, "*s*", onto the end of every strand in the tube *T*. It is denoted by Append (*T*, *s*).
5. *Append-head*: Given a tube *T* and a short strand of DNA, "*s*", the operation will append the short strand, "*s*", onto the head of every strand in the tube *T*. It is denoted by Append-head (*T*, *s*).
6. *Detect*: Given a tube *T*, say 'yes' if *T* includes at least one DNA molecule, and say 'no' if it contains none. It is denoted by Detect (*T*).
7. *Discard*: Given a tube *T*, the operation will discard the tube *T*. It is denoted by Discard (*T*).
8. *Read*: Given a tube *T*, the operation is used to describe a single molecule, contained in the tube *T*. Even if *T* contains many different molecules each encoding a different set of bases, the operation can give an explicit description of exactly one of them. It is denoted by Read (*T*).

## III. THE DNA ALGORITHMS FOR SOLVING THE SUBSET PRODUCT PROBLEM

### A. Definition of the Subset Product Problem

Assume that a finite set $S$ is $\{s_1, s_2,\ldots, s_q\}$, where $s_m$ is the $m$th element for $1\leq m \leq q$. Also suppose that every element in $S$ and a constant $M$ are a positive integer. The subset product problem is to determine whether there is a subset $S' \subseteq S$ such that the product of the sizes of the element in $S'$ is exactly $M$? The subset product problem has been proved to be the NP-complete problem [25]-[26].

Consider that a finite set $S$ = {1, 2, 3} and $M$=6. The total subsets for $S$ are, respectively, ∅, {1}, {2}, {3}, {1, 2}, {2, 3}, {1, 3} and {1,2,3}. According to the definition above of the Subset Product problem, the product of each subset is subsequently 0,1,2,3,2,6,4 and 6. So the solution of the subset product problem for $S$ is {2, 3} and {1, 2, 3}.In other word, the product of element in $S$={2, 3}or in $S$={1,2,3} is equal to constant value 6.

### B. Construct the Solution Space of DNA Strands for the Subset product Problem

Assume that $x_q x_{q-1}\ldots x_2 x_1$ is a $q$-bit binary number, is encoded to represent one of the $2^q$ subsets of a $q$-element set $S$. From [18, 19], for every bit $x_k$ representing the $k$th element in $S$ for $1\leq k \leq q$, two distinct 15-base value sequences are designed. One represents the value "0" for $x_k$ and the other represents the value "1" for $x_k$. For the sake of convenience in our representation, assume that $x_k^1$, which represents the $k$th element in set $S$ belongs to $S'$, denotes the value of $x_k$ to be 1 and $x_k^0$, which represents the $k$th element in set $S$ does not belong to $S'$, denotes the value of $x_k$ to be 0. The following DNA-based algorithm is used to construct the solution space for $2^q$ possible subsets of a $q$-element set $S$ in the subset product problem.

**Procedure Init** ($T_0$, $q$)
(1) Append ($T_1$, $x_q^1$).
(2) Append ($T_2$, $x_q^0$).
(3) $T_0 = \cup(T_1, T_2)$.
(4) **For** $k$= $q$ –1 **downto** 1
(4a) Amplify ($T_0$, $T_1$, $T_2$)
(4b) Append ($T_1$, $x_k^1$)
(4c) Append ($T_2$, $x_k^0$)
(4d) $T_0$= $\cup(T_1, T_2)$
**End For**
**EndProcedure**

The result generated by **Init**($T_0$, $q$) is shown as follows:

| Tube | The result generated by **Init** ($T_0$, $q$) |
|---|---|
| $T_0$ | $\{x_3^1x_2^1x_1^1, x_3^1x_2^1x_1^0, x_3^1x_2^0x_1^1, x_3^1x_2^0x_1^0, x_3^0x_2^1x_1^1, x_3^0x_2^1x_1^0, x_3^0x_2^0x_1^1, x_3^0x_2^0x_1^0\}$ |
| $T_1$ | ∅ |
| $T_2$ | ∅ |

**Lemma 1:** The algorithm, **Init** ($T_0$, $q$), is used to construct the solution space of $2^q$ possible subsets for a $q$-element set $S$.

### C. The Encoded Value for Every Element of Each Subset of a Finite Set

For the purpose of appending the DNA strands that encode the size of the element $s_m$ in each subset $S'$, we convert the element $s_m$ for $1\leq m \leq q$ representing the size of the element $m$ as a $2n$-bit binary number, $s_{m,2n}\, s_{m,2n-1},\ldots, s_{m,2}\, s_{m,1}$. Suppose that $s_{m,2n}$ is the most significant bit and $s_{m,1}$ is the least significant bit. For every bit $s_{m,k}$ , $1\leq m \leq q$ and $1 \leq k \leq 2n$, from [18]-[19] two *distinct* DNA sequences are designed. One corresponds to the value "0" for $s_{m,k}$ and the other corresponds to the value "1" for $s_{m,k}$. For the sake of convenience in our representation, we assume that $s_{m,k}^1$ denotes the value of $s_{m,k}$ to be 1 and $s_{m,k}^0$ defines the value of $s_{m,k}$ to be 0. The following algorithm is employed to construct the binary value of the elements in the $2^q$ subsets of a $q$-element set $S$.

**Procedure EncodedValue** ($T_0$, $q$, $2n$)
**(1)For** $m$ = 1 **to** $q$
(1a) $T_1$= + ($T_0$, $x_m^1$) and $T_2$= – ($T_0$, $x_m^1$)
(1b) **For** $k$ = $2n$ **downto** $n$+1
(1c) Append ($T_1$, $s_{m,k}$)
**End For**
(1d) **For** $k$ = $n$ **downto** 1
(1e) Append ($T_1$, $s_{m,k}^0$)
**End For**
(1f) Append ($T_2$, $s_{m,n+1}^1$)
(1g) **For** $k$ =$2n$ **downto** $n$+2
(1h) Append ($T_2$, $s_{m,k}^0$)
**End For**
(1i) **For** $k$ = $n$ **downto** 1
(1j) Append ($T_2$, $s_{m,k}^0$)
**End For**
(1k) $T_0$= $\cup(T_1, T_2)$
**EndFo**r
**EndProcedure**

**Lemma 2:** The binary value of the size for each element in $2^q$ subsets of a $q$-element set $S$ can be constructed from the algorithm, **EncodedValue**($T_0$, $q$, $n$).

### D. The Construction of a Parallel One-bit Adder

A one-bit adder is a Boolean function that forms the arithmetic sum of three inputs. It includes three inputs and two outputs. Two of the input bits represent augend and addend, respectively. The third input represents the carry from the previous lower significant position. The first output gives the value of the least significant bit of the sum for augend, addend and previous carry. The second output gives the output carry transferred into the input carry of the next one-bit adder. The truth table of the one-bit adder is shown in Table I.

Suppose that two one-bit binary numbers, $\alpha_{m-1,k}$ and $\alpha_{m,k}$, represent the first input (addend) and the first output (sum) of a one-bit adder for $1 \le m \le q$ and $1 \le k \le 2n$, respectively. A one-bit binary number, $\beta_{m,k}$, is applied to represent the second input(augend) of a one-bit adder. Two one-bit binary numbers, $\gamma_{m,k-1}$ and $\gamma_{m,k}$, are used to represent the third input (previous carry) and the second output (carry) of a one-bit adder respectively. From [18]-[19], two *distinct* DNA sequences are designed to represent the value "0" and "1" for every corresponding bit. For the sake of convenience in our representations, assume that $\beta_{m,k}^{1}$ contains the value of $\beta_{m,k}$ to be 1 and $\beta_{m,k}^{0}$ contains the value of $\beta_{m,k}$ to be 0. Also suppose that $\alpha_{m-1,k}^{1}$ denotes the value of $\alpha_{m-1,k}$ to be 1 and $\alpha_{m-1,k}^{0}$ defines the value of $\alpha_{m-1,k}$ to be 0. Similarly, suppose that $\alpha_{m,k}^{1}$ contains the value of $\alpha_{m,k}$ to be 1 and $\alpha_{m,k}^{0}$ denotes the value of $\alpha_{m,k}$ to be 0. $\gamma_{m,k-1}^{1}$ denotes the value of $\gamma_{m,k-1}$ to be 1 and $\gamma_{m,k-1}^{0}$ contains the value of $\gamma_{m,k-1}$ to be 0. $\gamma_{m,k}^{1}$ defines the value of $\gamma_{m,k}$ to be 1 and $\gamma_{m,k}^{0}$ contains the value of $\gamma_{m,k}$ to be 0. The following algorithm is proposed to perform the Boolean function of a parallel one-bit adder.

TABLE I
THE TRUTH TABLE OF THE ONE-BIT ADDER

| Augend bit | Addend bit | Previous Carry bit | Sum bit | Carry bit |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Procedure ParallelOneBitAdder**($T_0$, $\alpha_{m-1,k}$, $\beta_{m,k}$, $\gamma_{m,k-1}$, $m$, $k$)
(1) $T_1$= + ($T_0$, $\alpha_{m-1,k}^{1}$) and $T_2$= –( $T_0$, $\alpha_{m-1,k}^{1}$)
(2) $T_3$= + ($T_1$, $\beta_{m,k}^{1}$) and $T_4$= – ($T_1$, $\beta_{m,k}^{1}$)
(3) $T_5$= + ($T_2$, $\beta_{m,k}^{1}$) and $T_6$= – ($T_2$, $\beta_{m,k}^{1}$)
(4) $T_7$= + ($T_3$, $\gamma_{m,k-1}^{1}$) and $T_8$= – ($T_3$, $\gamma_{m,k-1}^{1}$)
(5) $T_9$= + ($T_4$, $\gamma_{m,k-1}^{1}$) and $T_{10}$= – ($T_4$, $\gamma_{m,k-1}^{1}$)
(6) $T_{11}$= + ($T_5$, $\gamma_{m,k-1}^{1}$) and $T_{12}$= – ($T_5$, $\gamma_{m,k-1}^{1}$)
(7) $T_{13}$= + ($T_6$, $\gamma_{m,k-1}^{1}$) and $T_{14}$= – ($T_6$, $\gamma_{m,k-1}^{1}$)
(8) **If** (Detect ($T_7$) = ="yes") **then**
Append-head ($T_7$, $\alpha_{m,k}^{1}$) and Append-head ($T_7$, $\gamma_{m,k}^{1}$)
**EndIf**
(9) **If** (Detect ($T_8$) = ="yes") **then**
Append-head ($T_8$,$\alpha_{m,k}^{0}$) and Append-head ($T_8$, $\gamma_{m,k}^{1}$)
**EndIf**
(10) **If** (Detect ($T_9$) = ="yes") **then**
Append-head ($T_9$,$\alpha_{m,k}^{0}$) and Append-head ($T_9$, $\gamma_{m,k}^{1}$)
**EndIf**
(11) **If** (Detect ($T_{10}$) = ="yes") **then**
Append-head ($T_{10}$, $\alpha_{m,k}^{1}$) and Append-head ($T_{10}$,$\gamma_{m,k}^{0}$)
**EndIf**
(12) **If** (Detect ($T_{11}$) = ="yes") **then**
Append-head ($T_{11}$, $\alpha_{m,k}^{0}$) and Append-head ($T_{11}$, $\gamma_{m,k}^{1}$)
**EndIf**
(13) **If** (Detect ($T_{12}$) = ="yes") **then**
Append-head ($T_{12}$, $\alpha_{m,k}^{1}$) and Append-head ($T_{12}$,$\gamma_{m,k}^{0}$)
**EndIf**
(14) **If** (Detect ($T_{13}$) = ="yes") **then**
Append-head ($T_{13}$, $\alpha_{m,k}^{1}$) and Append-head ($T_{13}$, $\gamma_{m,k}^{0}$)
**EndIf**
(15) **If** (Detect ($T_{14}$) = ="yes") **then**
Append-head ($T_{14}$, $\alpha_{m,k}^{0}$) and Append-head ($T_{14}$, $\gamma_{m,k}^{0}$)
**EndIf**
(16) $T_0 = \cup(T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14})$
**EndProcedure**

**Lemma 3:** The algorithm, **ParallelOneBitAdder** ($T_0$, $\alpha_{m-1,k}$, $\beta_{m,k}$, $\gamma_{m,k-1}$, $m$, $k$), can be applied to perform the Boolean function of a parallel one-bit adder.

*E. The Construction of a Parallel N-bit Adder Solution*

The parallel one-bit adder introduced in Section D figures out the arithmetic sum of two bits and a previous carry. Similarly, A binary parallel 2*n*-bit adder is also directly to perform the arithmetic sum for the two input operands of 2*n*-bit and the input carry by means of performing this one-bit adder 2*n* times. The following algorithm is proposed to perform the arithmetic sum for a parallel 2*n*-bit adder.

**Procedure ParallelAdder** ($T_0$, $\alpha$, $\beta$, $\gamma$, $q$, $2n$)
(1) **For** $k$ = 2$n$ **downto** 1
(1a) Append ($T_0$, $\alpha_{0,k}^{0}$)
**EndFor**
(2) **For** $m$ =1 **to** $q$
(2a) Append ($T_0$, $\gamma_{m,0}^{0}$)
(2b) **For** $k$ =1 **to** 2$n$
(2c) **ParallelOneBitAdder** ($T_0$, $\alpha_{m-1,k}$, $\beta_{m,k}$, $\gamma_{m,k-1}$, $m$, $k$)
**EndFor**
**EndFor**
**EndProcedure**

**Lemma 4:** The algorithm, **ParallelAdder**($T_0$, $\alpha$, $\beta$, $\gamma$, $q$, 2$n$), can be applied to perform the Boolean function to a binary parallel adder of 2*n* bits.

*F. The Construction of Right Shifter on DNA Computing*

The right shifter is applied to shift *A* register to the right, where *A* is a unsigned integers of *2n* bits. A symbol ">>" is used to represent the operation of right shift. Here the right shift must leave the sign unchanged. Assume that *A* can be represented as $A_{d,2n}$, $A_{d,2n-1}$, …, $A_{d,1}$ for $0 \le d \le 1$, where the value for $A_{d,k}$ for $1 \le k \le 2n$ is 1 or 0. Suppose that the *original* value for *A* is represented as $A_{0,2n}$, $A_{0,2n-1}$, …, $A_{0,1}$, the intermediate value of the first right shift for *A* is represented as $A_{1,2n}$, $A_{1,2n-1}$, …, $A_{1,1}$. In the processing of performing shift right, the bit $A_{0,1}$ is shifted out and is discarded, the bit $A_{1,2n}$ is filled with a bit "0".The content of

$A_{0,k}$ is shifted into the position of $A_{0,k-1}$, where $2 \le k \le 2n$. The truth table for ">>" (a right shifter) is shown as follows:

| Input | Output |
|---|---|
| $A_{d-1,k}$ | $A_{d,k-1}$ |
| 0 | 0 |
| 1 | 1 |

For the sake of convenience, $A_{d,k}{}^1$ for $0 \le d \le 1$ and $1 \le k \le 2n$ is that the value of $A_{d,k}$ is 1 and $A_{d,k}{}^0$ for $0 \le d \le 1$and $1 \le k \le 2n$ is that the value of $A_{d,k}$ is 0. The following algorithm is proposed to perform the parallel right shifter.

**Procedure ParallelRightShifter**($T_0, A$)
(1) Append-head($T_1$, $A_{0,2n}{}^1$).
(2) Append-head($T_2$, $A_{0,2n}{}^0$).
(3) $T_0 = \cup(T_1, T_2)$.
(4) **For** $k = 2n - 1$ **to** 1
(4a) Amplify($T_0$, $T_1$, $T_2$).
(4b) Append($T_1$, $A_{0,k}{}^1$).
(4c) Append($T_2$, $A_{0,k}{}^0$).
(4d) $T_0 = \cup(T_1, T_2)$.
**EndFor**
(5) **For** $d = 1$ **to** 1
(5a) Append($T_0$, $A_{d,2n}{}^0$).
(6) **For** $k = 2n$ **downto** 2
(6a) $T_3 = +(T_0, A_{d-1,k}{}^1)$ and $T_4 = -(T_0, A_{d-1,k}{}^1)$.
(6b) **If** (Detect($T_3$) = = "yes") **Then**
(6c) Append($T_3$, $A_{d,k-1}{}^1$).
**EndIf**
(6d) **If** (Detect($T_4$) = = "yes") **Then**
(6e) Append($T_4$, $A_{d,k-1}{}^0$).
**EndIf**
(6f) $T_0 = \cup(T_3, T_4)$.
**EndFor**
**EndFor**
**EndProcedure**

**Lemma 5**: The algorithm, **ParallelRightShifter**($T_0$,$A$), can be employed to carry out the parallel right shifter.

*G. The Construction of a parallel binary multiplier*

Multiplication of two fixed-point binary number in unsigned representation is done by successive additions and shifting. This process is illustrated with the following numerical example. Let us multiply the two binary numbers 0010 and 0011:

```
2        0010     Multiplicand
      X
3        0011     Mulitiplier
         0010
        0010
       0000
      0000
6     0000110     Product
```

When the above process is implemented in a DNA computing environment, it is better to change the process slightly. Our fast algorithm combines the rightmost half of the product with the multiplier. The algorithm starts by assigning the multiplier to the right half of the product register, placing 0 in the upper half. Depending on the value of the least significant bit of product is one or zero; we take the different operation steps in the procedure. The detailed iterations are shown as follows:

| Iteration | Step | Multiplicand | Product |
|---|---|---|---|
| 0 | Initial Values | 0010 | 0000 0011 |
| 1 | 1a: 1⇒Prod=Prod+ Mcand | 0010 | 0010 0011 |
| | 2: Shift right Product | 0010 | 0001 0001 |
| 2 | 1a: 1⇒Prod=Prod+ Mcand | 0010 | 0011 0001 |
| | 2: Shift right Product | 0010 | 0001 1000 |
| 3 | 1: 0⇒No Operation | 0010 | 0001 1000 |
| | 2: Shift right Product | 0010 | 0000 1100 |
| 4 | 1: 0⇒No Operation | 0010 | 0000 1100 |
| | 2: Shift right Product | 0010 | 0000 0110 |

In fact, if we ignore the signed bits, the length of an $n$-bit multiplicand and $m$-bits multiplier is a product that is $n + m$ bits long. A binary multiplier is a Boolean function that performs the arithmetic multiplication for the multiplicand and multiplier with the same length $n$ bits here. Therefore the variable length of product is $2n$ bits. The following algorithm is proposed to perform the Boolean function of a parallel binary multiplier.

**Procedure ParallelMultiplier**($T_0$, *Mcand*, *Mlier*, *Prod*, *q*, 2*n*)
/*initialization of product
(1) **For** $k = 2n$ **downto** $n+1$
Append ($T_0$, $Prod_{0,k}{}^0$).
**End for**
(2) **For** $k = n$ **to** 1
Append ($T_0$, $Mlier_{0,k}$).
**End for**
(3) **For** $k = 1$ **to** $n$
$T_3 = +(T_0, Prod_{0,1}{}^1)$ and $T_4 = -(T_0, Prod_{0,1}{}^1)$
**IF** (Detect($T_3$)='yes') **then**
**ParallelAdder**($T_0$, *Mlier*, Mcand, γ, *q*, *2n*)
**ParallelRightShifter**($T_0$,*Prod*)
**EndIf**
**IF** (Detect($T_4$)='yes') **then**
**ParallelRightShifter**($T_0$,*Prod*)

**EndIf**
**EndFor**
**EndProcedure**

**Lemma 6**: The algorithm, **ParallelMultiplier**($T_0$, *Mcand*, *Mlier*, *Prod*, *q*, 2*n*), can be employed to carry out the Boolean function of a parallel binary multiplier.

*H. Encode Any Given Positive Integer as a binary number in the Subset Product Problem*

Any given positive integer, *M*, can be converted as 2*n* one-bit binary numbers, $M_{2n}M_{2n-1}...M_2M_1$. The main advantage is that it is feasible for bit operations of the DNA algorithm in a parallel XOR operation. **Cons tant**($T_M$, 2*n*) is proposed to construct DNA strands for encoding *M*. The procedure is shown as follows:

**Procedure Constant** ($T_M$, 2*n*)
(1) **For** $k$ = 2*n* **downto** 1
(1a) Append ($T_M$, $M_k$)
**EndFor**
**EndProcedure**

**Lemma 7**: Any given positive integer *M* can be constructed with DNA strands from the algorithm, **Constant** ($T_M$, 2*n*).

*I. Constructing the Parallel One-bit XOR Operation on Bio-molecular Computing*

The Exclusive-OR (XOR) operation of a bit for two Boolean variables *A* and *B* generates an output of 1 if both *A* and *B* have different values and 0 if they are equal. The ⊕ symbol represents the XOR operation. The four possible combinations for the XOR operation of a bit with variables *A* and *B* are 0 ⊕ 0 = 0, 0 ⊕ 1 = 1, 1 ⊕ 0 = 1, 1 ⊕ 1 = 0. A truth table is usually used with logic operation to represent all possible combinations of inputs and the corresponding outputs. The truth table for the XOR operation is shown as follows:

| Input | | output |
|---|---|---|
| A | B | C = A ⊕ B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Assume that two one-bit binary numbers, $A_{q,k}$ and $B_{q,k}$ for $1 \le k \le 2n$, are used to respectively represent the first input and the second input for the XOR operation of a bit. Also suppose that a one-bit binary number, $C_{q,k}$ for $1 \le k \le 2n$, is employed to represent the output for the XOR operation of a bit. For the sake of convenience, assume that $A_{q,k}{}^1$ denotes that the value of $A_{q,k}$ is 1 and $A_{q,k}{}^0$ denotes that the value of $A_{q,k}$ is 0. Similarly, suppose that $B_{q,k}{}^1$ denotes that the value of $B_{q,k}$ is 1 and $B_{q,k}{}^0$ denotes that the value of $B_{q,k}$ is 0. Assume that $C_{q,k}{}^1$ denotes that the value of $C_{q,k}$ is 1 and $C_{q,k}{}^0$ denotes that the value of $C_{q,k}$ is 0. The following procedure is proposed to perform the parallel one-bit XOR operation.

**Procedure ParalleOneBitXOR**($T_0$, $A_{q,k}$, $B_{q,k}$, *q*, *k*)
(1) $T_1 = +(T_0, A_{q,k}{}^1)$ and $T_2 = -(T_0, A_{q,k}{}^1)$
(2) $T_3 = +(T_1, B_{q,k}{}^1)$ and $T_4 = -(T_1, B_{q,k}{}^1)$
(3) $T_5 = +(T_2, B_{q,k}{}^1)$ and $T_6 = -(T_2, B_{q,k}{}^1)$
(4) **If** (Detect($T_3$) = = "yes") **then**
(4a) Append-head($T_3$, $C_{q,k}{}^0$)
**EndIf**
(5) **If** (Detect($T_4$) = = "yes") **then**
(5a) Append-head ($T_4$, $C_{q,k}{}^1$)
**EndIf**
(6) **If** (Detect($T_5$) = = "yes") **then**
(6a) Append-head($T_5$, $C_{q,k}{}^1$)
**EndIf**
(7) **If** (Detect($T_6$) = = "yes") **then**
(7a) Append-head($T_6$, $C_{q,k}{}^0$)
**EndIf**
(8) $T_0 = \cup(T_3, T_4, T_5, T_6)$
**EndProcedure**

**Lemma 8**: The procedure, **ParallelOneBitXOR**($T_0$, $A_{q,k}$, $B_{q,k}$, *q*, *k*), can be employed to finish the parallel XOR one-bit Operation.

*J. Constructing the Parallel 2N-Bit XOR Operation on Bio-molecular Computing*

The parallel XOR operation of 2*n* bits simultaneously generates the corresponding 2*n*-bit outputs for XOR operation with two 2*n*-bit Boolean variables *A*, represented by $A_{q,2n}A_{q,2n-1}...A_{q,2}A_{q,1}$, and *B*, represented by $B_{q,2n}B_{q,2n-1}...B_{q,2}B_{q,1}$. The following procedure is offered to perform the parallel 2*n*-bit XOR operation for $2^q$ subsets of a *q*-element set *S*.

**Procedure ParallelXOR**($T_0$, *A*, *B*, *q*, 2*n*)
(1) **For** $k$ = 1 **to** 2*n*
(1a) **ParallelOneBitXOR**($T_0$, $A_{q,k}$, $B_{q,k}$, *q*, *k*).
**EndFor**
**EndProcedure**

**Lemma 9**: The procedure, **ParallelXOR**($T_0$, *A*, *B*, *q*, 2*n*), can be used to finish the parallel XOR operation of 2*n* bits.

*K. A DNA Algorithm for Solving the Subset Product Problem*

The following algorithm is used to solve the subset product problem of a *q*-element set *S*. Notations used in the following algorithm are denoted in the previous sections.

**Algorithm 1**: Solving the Subset Product Problem.
(1) **Init**($T_0$, *q*)
(2) **EncodedValue**($T_0$, *q*, 2*n*)
(3) **ParallelMultiplier** ($T_0$, *Mcand*, *Mlier*, *Prod*, *q*, 2*n*)
(4) **Constant** ($T_M$, 2*n*)
(5) **ParallelXOR**($T_0$, *Prod*, *M*, *q*,2 *n*)
(6) **For** $k$ = 1 **to** 2*n*
(6a) $T_1 = +(T_0, C_{q,k}{}^0)$ and $T_2 = -(T_0, C_{q,k}{}^0)$
(6b) $T_0 = \cup(T_1, T_0)$
(6c) Discard ($T_2$)
**EndFor**
(7) **If** (Dectect ($T_0$) = = "yes") **then**

(7a) Read($T_0$)
**EndIf**
**EndAlgorithm**

**Theorem 1**: From those steps in **Algorithm 1**, the subset product problem for $2^q$ subsets of a $q$-element set $S$ can be solved.

*L. The Complexity of Algorithm 1*

**Theorem 2:** Suppose that a finite set $S$ is $\{s_1, s_2, \ldots, s_q\}$. The subset product problem for $S$ can be solved with O ($q \times n$) biological operations, O ($2^n$) library strands, O (1) tubes and the longest library strands, O ($q \times n$), where $2n$ is the number of bits for representing the value of each element in $S$.

## IV. CONCLUSIONS

The subset product problem proved to be the NP-complete problem by restriction has been solved by a number of different algorithms with exponential time complexity in conventional silicon-based computer [26]. Here the proposed algorithm for solving the subset product problem is based on basic biological operations. The number of tubes, the number of biological operations, the number of memory strands and the longest length of memory strands, respectively, are O (1), O ($q \times n$), O ($2^n$) and O ($q \times n$).

The presented algorithm has several advantages lying in its massive parallelism as described below. First, these biological operations needed in the proposed algorithm are experimentally feasible in lab level [18]-[19]. Second, the DNA-based algorithms of polynomial time complexity are proposed for searching the subset product problem. Third, the Adleman program [18]-[19] can be applied to generate good DNA sequences for the constructing solution space of our problem. It demonstrates that the proposed algorithm has a lower rate of errors for hybridization. Fourth, the contribution of this study is that DNA-based algorithms developed herein can be applied to solve addition-related problems.

## References


[1] R. P. Feynman, "In miniaturization," D.H. Gilbert, Ed., Reinhold Publishing Corporation, New York, 1961, pp. 282-296.

[2] L. Adleman, "Molecular computation of solutions to combinatorial problems," Science, 266, November 11, 1994, pp. 1021-1024

[3] R. J. Lipton, "DNA solution of hard computational problems," Science, 268, 1995,pp. 542:545

[4] S. Roweis, E. Winfree, R. Burgoyne, N. V. Chelyapov, M. F. Goodman, Paul W.K. Rothemund, and L. M. Adleman, "A sticker based model for DNA computation," 2nd annual workshop on DNA Computing, Princeton University. Eds. L. Landweber and E. Baum, DIMACS: series in *Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, *1999*, pp. 1-29.

[5] M. Guo, W. L. Chang, M. Ho, J. Lu and J. Cao. "Is optimal solution of every NP-complete or NP-hard problem determined from its characteristic for DNA-based computing". *Biosystems*, Vol. 80, No. 1, pp. 71-82, 2005.

[6] Q. Quyang, P.D. Kaplan, S. Liu, and A. Libchaber. "DNA solution of the maximal clique problem". Science, 278:446-449, 1997.

[7]M. Amos. "DNA Computation", Ph.D. Thesis, department of computer science, the University of Warwick, 1997.

[8] W. L. Chang, M. Guo, and M. Ho, "Towards solution of the set-splitting problem on gel-based DNA computing", *Future Generation Computer Systems*, Volume: 20, Issue: 5, June 15, 2004, pp. 875-885.

[9] W. L. Chang, M. Ho, and M. Guo, "Fast parallel molecular solution to the dominating-set problem on massively parallel bio-computing," *Parallel Computing (Elsevier Science)*, Vol. 30, No. 9&10, 2004, pp. 1109-1125.

[10] D. Xiao, W. Li, Z. Zhang, and L. He. "Solving the maximum cut problems in the Adleman–Lipton model". *BioSystems*, Volume 82, pp. 203-207, 2005.

[11] C. W. Yeh, C. P. Chu and K. R. Wu. "Molecular solutions to the binary integer programming problem based on DNA computation". *Biosystems*, Volume: 83, Issue: 1, January, 2006, pp. 56-66.

[12] D. Boneh, C. Dunworth, and R. J. Lipton. "Breaking DES using a molecular computer". In Proceedings of the 1*st* DIMACS Workshop on DNA Based Computers, 1995, American Mathematical Society. In DIMACS Series in *Discrete Mathematics and Theoretical Computer Science*, Volume 27, pp. 37-66, 1996.

[13] L. Adleman, P. W. K. Rothemund, S. Roweis, and E. Winfree. "On applying molecular computation to the Data Encryption Standard". The 2*nd* annual workshop on DNA Computing, Princeton University, DIMACS: series in *Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, pp. 31-44, 1999.

[14] F. Guarnieri, M. Fliss, and C. Bancroft, "Making DNA add," *Science*, Vol. 273, pp. 220–223, 1996.

[15] H. Ahrabian and A. Nowzari-Dalini. "DNA simulation of nand Boolean circuits". Advanced Modeling and Optimization, Volume 6, Number 2, 2004, pp. 33-41.

[16] A. Schuster. "DNA databases". *BioSystems*, Volume 81, 2005, pp. 234–246.

[17] W.L. Chang. "Fast Parallel DNA-based Algorithms for Molecular Computation: the Set-Partition Problem". *IEEE Transactions on Nanobioscience*, Vol. 6, No. 1, 2007, pp 346 - 353.

[18 ] R. S. Braich, C. Johnson, P. W.K. Rothemund, D. Hwang, N. Chelyapov, and L. M. Adleman. "Solution of a satisfiability problem on a gel-based DNA computer". Proceedings of the 6th International Conference on DNA Computation in the Springer-Verlag Lecture Notes in Computer Science series, pp 1-27, 2001.

[19] L. M. Adleman, R. S. Braich, C. Johnson, P. W. K. Rothemund, D. Hwang and N. Chelyapov. "Solution of a 20-Variable 3-SAT Problem on a DNA Computer". *Science*, Volume 296, Issue 5567, 499-502, 19 April, 2002.

[20] J. Watson, M. Gilman, J. Witkowski, and M. Zoller. *Recombinant DNA* (2*nd* edition). Scientific American Books, W. H. Freeman and Co., 1992.

[21] J. Watson, N. Hoplins, J. Roberts, and et. al. *Molecular Biology of the Gene*. Benjamin/Cummings Menlo Park CA, 1987.

[22] G. M. Blackburn and M. J. Gait. *Nucleic Acids in Chemistry and Biology*. IRL Press, 1990.

[23] F. Eckstein. *Oligonucleotides and Anologues*. Oxford University Press. 1991.

[24] M. Amos. *Theoretical and Experimental DNA Computation*. Springer, 2005, ISBN 3540657738.

[25] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms* (the econd edition). MIT Press, 2001.

[26] M. R. Garey, and D. S. Johnson. *Computer and intractability*. Freeman, San Fransico, CA, 1979